数据库索引

索引是什么，有什么作用以及优缺点

索引是对数据

是帮助MySQL高效获取数据的数据结构。可以理解为索引就是加快检索表中数据的方法。数据库的索引类似于属鸡的索引。在书籍中，索引允许用户不必翻阅完整个书就能得到所需要的信息。在数据库中，索引也允许数据库程序迅速的找到表中的数据，而不息扫描整个数据库。

索引的优点

1 创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

2 可以大大加快数据检索的速度，这也是创建索引的最主要的原因。

3 可以加速表和表之间的链接。特别是在实现数据的参考完整性方面特别有意义。

4 在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

5 通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统性能。

索引的缺点

第一创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二 索引需要占用物理内存，除了数据表占用数据空间之外，每一个索引还要占一定的物理空间。

第三 当对表中的数据进行增删改的时候，索引也要动态维护，这样就降低了数据的维护速度。

为什么要给表加上主键？

我们平时建表的时候都会为表加上主键，在某些关系型数据库中，如果建表时不指定主键，数据库会拒绝建表的语句执行。事实上，一个加了主键的表，并不能称之为[表]。一个没加主键的表，它的数据无序的放置在磁盘存储器上，一行行的排列的很整齐，跟我们认知的表很接近。

如果给表加上主键，那么表在磁盘中存储的结构就由整齐排列的结构转变成了树状结构，也就是平衡树结构。换句话说，就是整个表变成了一个索引，也就是**聚集索引**。这就是为什么一个表只能有一个主键，一个表只能有一个聚集索引，因为主键的作用就是把表的数据格式转换成索引平衡树的格式放置。

B+数的**所有结点（底部除外）的数据**都是由**主键字段中的数据构成的**，也就是通常我们指定**主键的id字段**。最下面是真正表中的数据。假如我们执行一个SQL语句：

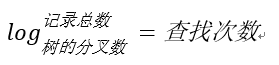
select \* from table where id = 1256;

首先根据索引定位到1256这个值所在的叶结点，然后再通过叶结点取到id等于1256的数据行。

为什么加了索引之后会使查询变快？

加入一张表有一亿个数据，需要查找其中的某一条数据，按照常规逻辑，一条一条的去匹配的话，最坏的情况下需要匹配一亿次才能得到结果，用大O标记法就是O(n)最坏时间幅度，这是无法接受的，而且这一亿条数据歉然不能一次性读入内存供程序使用。因此这一亿次匹配在不经缓存优化的情况下就是一亿次IO开销，以现在磁盘的IO能力和CPU运算能力，有可能需要几个月的时间才能得到结果。

如果把这张表转换成平衡树结构（一课非常茂盛和结点非常多的树），假设这棵树有10层，那么只需要10次IO开销就能查找到所需要的数据，速度以指数级别提升。用大O标记法就是O(logN)，N是记录总数，底数是树的分叉数结果就是树的层次数，查找次数就是以树的分叉数为底数记录总数的对数。用公式为



用程序来表示就是Math.Log(100000000,10)，100000000是记录数，10是树的分叉数（真实环境下分叉数远不止10）， 结果就是查找次数，这里的结果从亿降到了个位数。因此，利用索引会使数据库查询有惊人的性能提升。

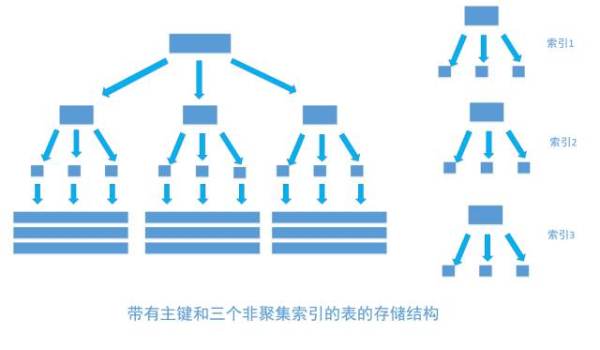
**为什么加了索引之后会使写入，修改、删除变慢？**

索引能够使数据库的查询速度上升，而写入速度下降，原因很简单。因为平衡树这和结构必须一直维持在一个正确的状态，增删改数据都会改变平衡树各节点的索引数据内容，破坏树的结构，因此每次数据改变时，DBMS必须重新梳理树（索引）的结构以确保它的正确。这会带来不小的性能开销，也就是为什么索引会给查询以外的操作带来副作用的原因。

**索引的分类 聚集索引和非聚集索引**

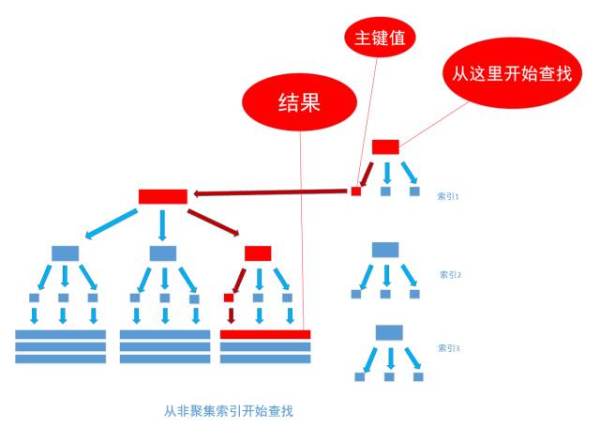
非聚集索引也就是我们经常提起和使用的常规索引。

非聚集索引和聚集索引一样，同样是采用平衡树作为索引的数据结构。**索引树中各节点的值来自于表中的索引字段**，假如给user表的name字段加上索引，那么索引就是有name字段中的值构成的，在数据改变时，DBMS需要一直维护索引结构的正确性。如果给表中的多个字段加上索引，那么就会出现多个独立的索引结构，**每个索引（非聚集索引）互相之间不存在关联。**



每次给字段建立一个新的索引，字段中的数据就会被复制一份出来，用于生成索引。因此给表添加索引，会增加表的体积，占用磁盘存储空间。

非聚集索引和聚集索引的区别在于，通过聚集索引可以查到需要查找的数据，而通过非聚集索引可以查找到记录对应的主键值。再使用主键值通过聚集索引查找到需要的数据。



不管以任何方式查询表，最终都会利用主键通过聚集索引来定位到数据，聚集索引（主键）是通往真实数据所在的唯一路径。

然而，有一种例外可以不使用聚集索引就能查询到所需要的数据，这种非主流的方法称之为**覆盖索引查询**，也就是复合索引或多字段索引查询。上面已经指出，当为字段建立索引以后，字段中的内容会被同步到索引中，如果为一个索引指定两个字段，那么这两个字段的内容都会被同步至索引中。

//建立索引

Create index index\_birthday on user\_innfo(birthday0;

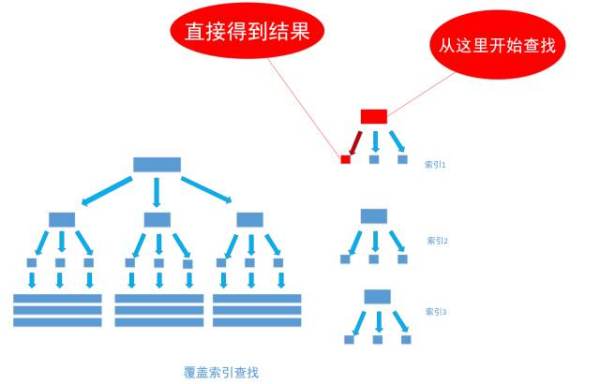
//查询生日在1991年11月1日出生的用户的用户名

Select user\_name from user\_info where birthday=’1991-11-1’

这句SQL语句的执行过程如下

首先，通过非聚集索引index\_birthday查找birthday等于1991-11-1的所有记录的主键ID值，然后，通过得到的主键ID值，执行聚集索引查找，找到主键ID值对应的直至数据（数据行）存储位置。

最后，从得到的真实数据中取得user\_name字段的值返回，也就是取得最终的结果。



如果将birthday字段上的索引改成双字段的覆盖索引

Create index index\_birthday\_and\_user\_name on user\_info(birthday,user\_name);

这句SQL语句的执行过程就会变成

通过非聚集索引index\_birthday\_and\_user\_name查找birthday等于1991-11-1的叶节点的内容，然而，叶节点中除了有user\_name表主键ID的值之外，user\_name字段的值也在里面，因此无需通过住家ID的值来查找数据行的真是所在。，直接取得叶节点中的uer\_name的值返回即可。通过这种覆盖索引直接查找方式，可以审略不适用堵盖索引查找后面的两个步骤，大大提高了查询的性能。